1 Overview

1.1 Description of the problem

This project implements deep learning, an artificial intelligence technique, to classify x-rays from patients to predict whether or not they have pneumonia. The dataset utilized in this project is from Kaggle, and can be found here: [https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia (https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia)](https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia)

In this dataset, there are over 5,800 images of x-rays from healthy patients and patients with pneumonia. The business case for establishing a robust predictive model relates to developing software to automatically read x-rays scan in patients and determine if they have pneumonia. This software could assist the efforts of radiologists or offer a better solution to diagnosing pneumonia.

1.2 How should results be evaluated?

A good goal for this project could be creating a model that is at least better than entry-level radiologists at diagnosing pneumonia. Achieving results better than this standard would allow for more promise in creating higher achieving models that ideally would trump the performance of any radiologist. According to [IBM (https://www.ibm.com/blogs/research/2020/11/ai-x-rays-for-radiologists/)](https://www.ibm.com/blogs/research/2020/11/ai-x-rays-for-radiologists/), the sensitivity, specificity, and positive predictive value for radiology residents is as follows: 0.720, 0.973, and 0.682 respectively.
It will be useful to review each of these metrics below.

Sensitivity and specificity are defined as the following:

$$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$

Let's review some definitions relating to this project.

- True Positive- Patient is predicted to have pneumonia and does have pneumonia.
- False Negative- Patient is predicted to be healthy but does have pneumonia.
- True Negative- Patient is predicted to be healthy and is healthy.

- False Positive- Patient is predicted to have pneumonia but is healthy.

Given these definitions, sensitivity relates to how many patients are accurately predicted to have pneumonia divided by the sum of this amount and patients that are predicted to be healthy but are not. Sensitivity indicates how many patients were accurately diagnosed out of the total amount of afflicted patients.

Specificity is the amount of patients accurately predicted to be healthy divided by the sum of this number and patients predicted to have pneumonia but are healthy. Specificity indicates how many healthy patients are accurately indicated divided by the total number of healthy patients. In the case of the radiology residents, they are quite good at predicting when the patient is healthy.

Positive predictive value is defined as the following:

$$\mathrm{PPV} = \frac{\text{Number of true positives}}{\text{Number of true positives} + \text{Number of false positives}}$$

Positive predictive value is equal to the amount of patients predicted to have pneumonia and have pneumonia divided by the sum of this value and the amount of patients without pneumonia that were predicted to have it. PPV is essentially referring to how many correct positive predictions we have divided by the total amount of positive predictions. The resident radiologists were the worst in this category, which indicates that they were not very accurate in reference to the total amount of cases that were diagnosed. Given all the patients diagnosed as having pneumonia, about 68.2% actually had the disease.

It should be noted that a false negative is certainly worse than a false positive. It seems obvious that a false negative (a patient that has pneumonia but is predicted as healthy) is quite bad because this patient and the healthcare professionals may assume they are healthy and will not pursue treatment, or they will pursue more healthcare such as additional tests. A false positive is significantly better because treating pneumonia typically entails prescribing antibiotics and sometimes other medicines like cough medicine, and there are typically not significant effects that can occur from a healthy person taking antibiotics.

1.3 Recap: Metrics used in this project

The metrics being used in this project along with the percent value of each metric signifying the goal to beat are as follows:

Metric	Score to Beat
Sensitivity	72.0%
Specificity	97.3%
Positive Predictive Value	68.2%

2 Exploratory Data Analysis (EDA)

2.1 Import packages and data

In [1]:
```
# Imports
import pandas as pd
import numpy as np
from pathlib import Path
import glob
import os
from os import listdir
from pathlib import Path

from skimage.io import imread
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn

%matplotlib inline

from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_i
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras import models, layers, optimizers, regularizers
from tensorflow.keras import activations
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.applications import VGG16, VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau

from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import plot_confusion_matrix

import datetime

from tensorflow.random import set_seed
set_seed(13)
```

```
C:\Users\18016\anaconda3\envs\learn-env\lib\site-packages\skimage\io\manage_plu
gins.py:23: UserWarning: Your installed pillow version is < 8.1.2. Several secu
rity issues (CVE-2021-27921, CVE-2021-25290, CVE-2021-25291, CVE-2021-25293, an
d more) have been fixed in pillow 8.1.2 or higher. We recommend to upgrade this
library.
  from .collection import imread_collection_wrapper
```

2.2 Helper functions

The following function is for labeling data from paths. This function is mainly to help with preparing the data for Exploratory Data Analysis (EDA)

In [2]:

```
# This function loads in data from two paths, assigns a label to the data:
# 0 for normal and 1 for pneumonia, and then combines the labeled data into
# dataframe
# Input: path1 (normal data), path2 (pneumonia data)
# output: dataframe
def label_data(path1, path2):
    # assign images in the paths to variables for later labeling assignment
    norm = path1.glob('*.jpeg')
    pn = path2.glob('*.jpeg')

    # holder for labeled images
    data = []

    # label normal images 0
    for img in norm:
        data.append((img,0))

    # label pneumonia as 1
    for img in pn:
        data.append((img, 1))

    # create dataframe with images and labels
    df = pd.DataFrame(data, columns=['image', 'pneumonia'], index=None)

    # randomize the data
    df = df.sample(frac=1, random_state=42).reset_index(drop=True)

    return df
```

Creating paths for the data and then using the above function to label.

In [3]:

```
# set paths for the data files and then label the data
train_path_norm = Path('data/train/NORMAL')
train_path_pn = Path('data/train/PNEUMONIA')

test_path_norm = Path('data/test/NORMAL')
test_path_pn = Path('data/test/PNEUMONIA')

val_path_norm = Path('data/val/NORMAL')
val_path_pn = Path('data/val/PNEUMONIA')

# label and create df for train, test, and validation datasets
df_train = label_data(train_path_norm, train_path_pn)
df_test = label_data(test_path_norm, test_path_pn)
df_val = label_data(val_path_norm, val_path_pn)
```

In [4]:

```python
# check to make sure the labeling function worked.
df_train.head()
```

Out[4]:

	image	pneumonia
0	data\train\PNEUMONIA\person1288_virus_2211.jpeg	1
1	data\train\NORMAL\NORMAL2-IM-0816-0001.jpeg	0
2	data\train\PNEUMONIA\person61_bacteria_290.jpeg	1
3	data\train\PNEUMONIA\person722_virus_1341.jpeg	1
4	data\train\PNEUMONIA\person1141_virus_1890.jpeg	1

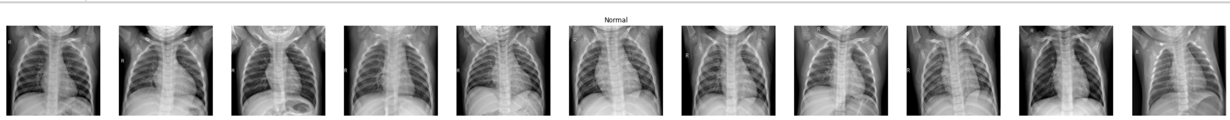
- The dataframe contains the image paths for each image along with the label for each image.

2.3 Initial look at the data

Examine images of both normal and pneumonia x-rays

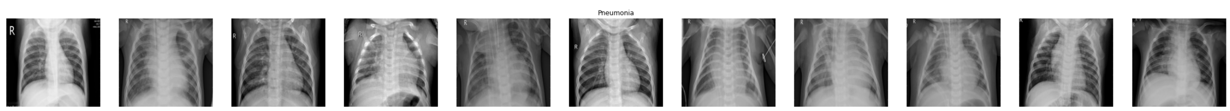
In [5]:

```python
# plot some images of some normal x-rays
n_img = (df_train[df_train['pneumonia']==0]['image'].iloc[:11]).tolist()

fig, ax = plt.subplots(1, 11, figsize=(40, 3))
for i in range(11):
    to_plot = imread(n_img[i])
    ax[i].imshow(to_plot, cmap='gray')
    ax[i].axis('off')
    ax[i].set_aspect('auto')
    if i == 5:
        ax[i].set_title('Normal')

plt.show()
```



Normal x-rays tend to be clear, and there is typically a prevalence of black space in the imagery.

In [6]:
```
# plot some images of some pneumonia x-rays
pn_img = (df_train[df_train['pneumonia']==1]['image'].iloc[:11]).tolist()

fig, ax = plt.subplots(1, 11, figsize=(40, 3))
for i in range(11):
    to_plot = imread(pn_img[i])
    ax[i].imshow(to_plot, cmap='gray')
    ax[i].axis('off')
    ax[i].set_aspect('auto')
    if i == 5:
        ax[i].set_title('Pneumonia')

plt.show()
```



There is typically more cloudiness and white areas in pneumonia cases, but it is difficult to tell for some cases (at least for an untrained person) in this set.

In [7]:
```
# look at shape of train, test, and validation
print(df_train.shape, df_test.shape, df_val.shape)
```

```
(5216, 2) (624, 2) (16, 2)
```

The train set has 5,216 observations, the test set has 624 observations, and the validation set only has 16 observations.

2.4 Observe distribution of classes in the data

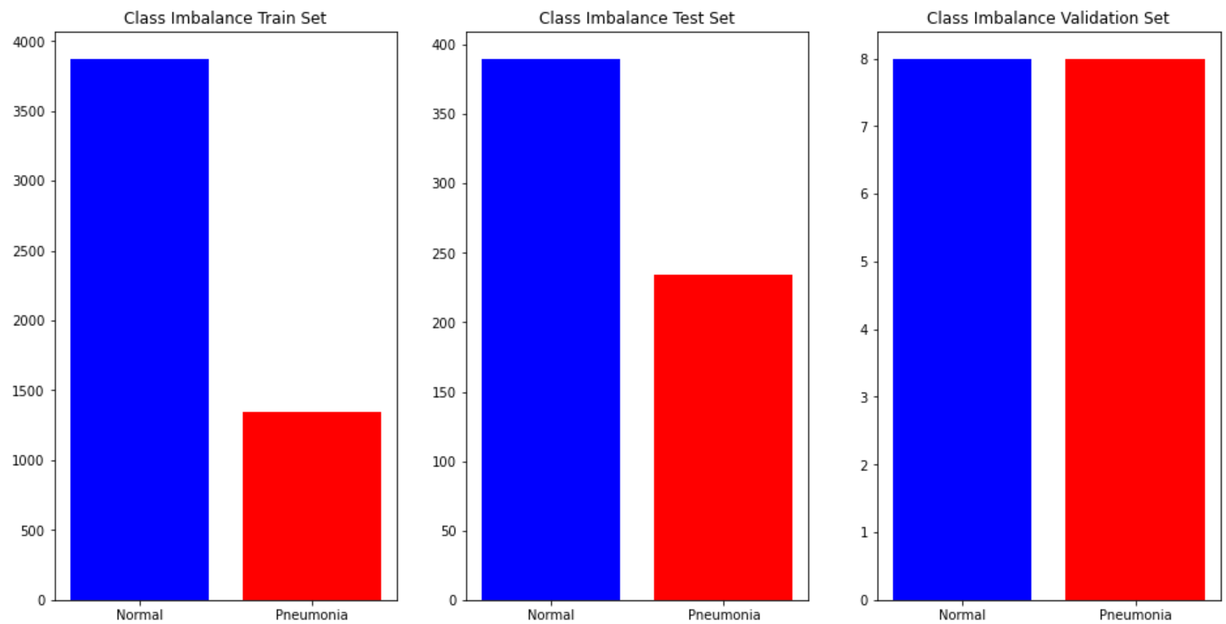
In [8]:
```
# look into class imbalance in each set
x1 = df_train['pneumonia'].value_counts()
x2 = df_test['pneumonia'].value_counts()
x3 = df_val['pneumonia'].value_counts()
print(x1)
print(x2)
print(x3)
```

```
1    3875
0    1341
Name: pneumonia, dtype: int64
1    390
0    234
Name: pneumonia, dtype: int64
1    8
0    8
Name: pneumonia, dtype: int64
```

- There are 3875 normal xrays and 1341 pneumonia xrays in train data
- There are 390 normal xrays and 234 pneumonia xrays in the test data
- There are 8 normal xrays and 8 pneumonia xrays in the validation data

In [9]:

```python
# visualize class imbalance in all three data sets
fig, ax = plt.subplots(1, 3, figsize=(16,8), sharey=False)
ax[0].bar(['Normal', 'Pneumonia'], [3875, 1341], color=['b', 'r'])
ax[0].set_title('Class Imbalance Train Set')

ax[1].bar(['Normal', 'Pneumonia'], [390, 234], color=['b', 'r'])
ax[1].set_title('Class Imbalance Test Set')

ax[2].bar(['Normal', 'Pneumonia'], [8, 8], color=['b', 'r'])
ax[2].set_title('Class Imbalance Validation Set')

plt.show()
```



- There is more class imbalance in the train set than the other two.
- There is equal class balance in the validation set, but there are only 16 total observations in the validation set.
- The class imbalance is not severe and probably does not to be accounted for in the modeling phase.

3 Modeling

3.1 Prepare images for modeling & feature engineering

Use ImageDataGenerator as a feature engineering method to alter the images. This process allows for reduction in bias that may be learned by future models. For example, pneumonia may present itself more prevalently in a particular region in the chest cavity in this dataset. The permutations found in ImageDataGenerator allow for a reduction in such biases.

In [10]:
```
# ImageDataGenerators for train, test, validation
# generate images
permutes = ImageDataGenerator(
    rescale = 1. / 255,      # multiply the data by the value provided
    shear_range = 0.2,      # this distorts the image along an axis
    zoom_range = 0.2,      # range for random zoom
    horizontal_flip = True      # random horizontal flip
)

# initialize variables to reduce redundancies
batch_size = 16      #  number of samples that the NN uses each iteration

train_gen = permutes.flow_from_directory(
    'data/train',
    target_size = (224, 224),
    batch_size = 5216,
    class_mode = 'binary'
)

test_gen = permutes.flow_from_directory(
    'data/test',
    target_size = (224, 224),
    batch_size = 624,
    class_mode = 'binary'
)

val_gen = permutes.flow_from_directory(
    'data/val',
    target_size = (224, 224),
    batch_size = batch_size,
    class_mode = 'binary'
)
```

```
Found 5216 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
```

Note the comments of each parameter in the ImageDataGenerator. Each one of these parameters distorts the images in the set in different ways, and the comments in the above code block explain how.

3.2 Train-Test-Validation datasets

In [11]:
```
# Create the data sets with the train/test/val splits
X_train, y_train = next(train_gen)
X_test, y_test = next(test_gen)
X_val, y_val = next(val_gen)
```

Preview an image in one of the sets

In [12]:
```
X_train[2]
```

Out[12]:
```
array([[[0.27450982, 0.27450982, 0.27450982],
        [0.27450982, 0.27450982, 0.27450982],
        [0.27450982, 0.27450982, 0.27450982],
        ...,
        [0.13125975, 0.13125975, 0.13125975],
        [0.13128105, 0.13128105, 0.13128105],
        [0.13130236, 0.13130236, 0.13130236]],

       [[0.27450982, 0.27450982, 0.27450982],
        [0.27450982, 0.27450982, 0.27450982],
        [0.27450982, 0.27450982, 0.27450982],
        ...,
        [0.12680155, 0.12680155, 0.12680155],
        [0.12679444, 0.12679444, 0.12679444],
        [0.12678733, 0.12678733, 0.12678733]],

       [[0.28108993, 0.28108993, 0.28108993],
        [0.28106862, 0.28106862, 0.28106862],
        [0.2810473 , 0.2810473 , 0.2810473 ],
        ...,
        [0.12816319, 0.12816319, 0.12816319],
        [0.12817737, 0.12817737, 0.12817737],
        [0.12819159, 0.12819159, 0.12819159]],

       ...,

       [[0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        ...,
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ]],

       [[0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        ...,
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ]],

       [[0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        ...,
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        ]]], dtype=float32)
```

- The data structure above represents an x-ray in the training set.
- The permuted values can be seen (i.e. rescaling)

3.3 Model 1: Baseline

Build a neural net with a few layers for the first model.

Binary cross entropy will be used as the loss function. It can be visualized below:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Put very simply, this formula is penalizing false predictions and rewarding true predictions. For a more detailed explanation of why this formula is used, refer to [this article (https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a)](https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a).

The optimizer used for the baseline model is Stochastic Gradient Descent (SGD). More on optimizers can be found [here (https://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent)](https://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent).

3.3.1 Create and fit model

In [13]:

```
# Baseline model
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(224, 224, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(
    loss='binary_crossentropy',     # loss function should be binary for 2
    optimizer='sgd',
    metrics=['acc']
)
```

In [14]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d (Conv2D)             (None, 222, 222, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 32)     0
_________________________________________________________________
conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 64)       0
_________________________________________________________________
conv2d_2 (Conv2D)           (None, 52, 52, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 26, 26, 64)       0
_________________________________________________________________
flatten (Flatten)           (None, 43264)             0
_________________________________________________________________
dense (Dense)               (None, 64)                2768960
_________________________________________________________________
dense_1 (Dense)             (None, 1)                 65
=================================================================
Total params: 2,825,345
Trainable params: 2,825,345
Non-trainable params: 0
_________________________________________________________________
```

In [15]:
```
original_start = datetime.datetime.now()
start = datetime.datetime.now()

history = model.fit(
    X_train,
    y_train,
    epochs=25,
    validation_data=(X_val, y_val)
)

end = datetime.datetime.now()
elapsed = end - start
print('Training took a total of {}'.format(elapsed))
```

```
Epoch 1/25
163/163 [==============================] - 67s 411ms/step - loss: 0.5324 - acc: 0.7446 - val_loss: 0.7806 - val_acc: 0.5625
Epoch 2/25
163/163 [==============================] - 67s 411ms/step - loss: 0.4167 - acc: 0.8069 - val_loss: 0.8262 - val_acc: 0.6250
Epoch 3/25
163/163 [==============================] - 66s 406ms/step - loss: 0.3057 - acc: 0.8729 - val_loss: 0.7177 - val_acc: 0.6875
Epoch 4/25
163/163 [==============================] - 65s 401ms/step - loss: 0.2670 - acc: 0.8909 - val_loss: 0.6331 - val_acc: 0.8750
Epoch 5/25
163/163 [==============================] - 66s 408ms/step - loss: 0.2525 - acc: 0.8957 - val_loss: 1.0871 - val_acc: 0.5000
Epoch 6/25
163/163 [==============================] - 68s 417ms/step - loss: 0.2365 - acc: 0.9051 - val_loss: 0.9061 - val_acc: 0.5625
Epoch 7/25
163/163 [==============================] - 70s 428ms/step - loss: 0.2262 - acc: 0.9053 - val_loss: 0.8195 - val_acc: 0.6250
Epoch 8/25
163/163 [==============================] - 70s 431ms/step - loss: 0.2195 - acc: 0.9112 - val_loss: 1.0671 - val_acc: 0.5625
Epoch 9/25
163/163 [==============================] - 70s 429ms/step - loss: 0.2086 - acc: 0.9187 - val_loss: 0.8841 - val_acc: 0.5625
Epoch 10/25
163/163 [==============================] - 70s 431ms/step - loss: 0.1992 - acc: 0.9212 - val_loss: 0.7607 - val_acc: 0.6250
Epoch 11/25
163/163 [==============================] - 70s 432ms/step - loss: 0.1952 - acc: 0.9264 - val_loss: 1.0659 - val_acc: 0.5625
Epoch 12/25
163/163 [==============================] - 66s 407ms/step - loss: 0.1837 - acc: 0.9314 - val_loss: 0.9283 - val_acc: 0.5625
Epoch 13/25
163/163 [==============================] - 65s 398ms/step - loss: 0.1796 - acc: 0.9314 - val_loss: 1.1906 - val_acc: 0.5000
Epoch 14/25
163/163 [==============================] - 65s 396ms/step - loss: 0.1737 - acc: 0.9360 - val_loss: 0.9702 - val_acc: 0.5625
Epoch 15/25
```

```
163/163 [==============================] - 65s 397ms/step - loss: 0.1652 - acc: 0.9377 - val_loss: 0.7560 - val_acc: 0.5000
Epoch 16/25
163/163 [==============================] - 65s 398ms/step - loss: 0.1614 - acc: 0.9388 - val_loss: 1.2316 - val_acc: 0.5000
Epoch 17/25
163/163 [==============================] - 67s 412ms/step - loss: 0.1586 - acc: 0.9419 - val_loss: 0.8280 - val_acc: 0.6250
Epoch 18/25
163/163 [==============================] - 68s 419ms/step - loss: 0.1531 - acc: 0.9450 - val_loss: 1.1766 - val_acc: 0.5625
Epoch 19/25
163/163 [==============================] - 67s 411ms/step - loss: 0.1462 - acc: 0.9446 - val_loss: 1.3945 - val_acc: 0.5625
Epoch 20/25
163/163 [==============================] - 66s 405ms/step - loss: 0.1416 - acc: 0.9496 - val_loss: 1.1174 - val_acc: 0.5625
Epoch 21/25
163/163 [==============================] - 67s 408ms/step - loss: 0.1415 - acc: 0.9492 - val_loss: 0.9514 - val_acc: 0.6250
Epoch 22/25
163/163 [==============================] - 66s 403ms/step - loss: 0.1349 - acc: 0.9511 - val_loss: 0.8218 - val_acc: 0.6250
Epoch 23/25
163/163 [==============================] - 65s 401ms/step - loss: 0.1311 - acc: 0.9544 - val_loss: 1.1999 - val_acc: 0.6875
Epoch 24/25
163/163 [==============================] - 66s 408ms/step - loss: 0.1283 - acc: 0.9513 - val_loss: 0.9382 - val_acc: 0.6875
Epoch 25/25
163/163 [==============================] - 67s 410ms/step - loss: 0.1246 - acc: 0.9553 - val_loss: 0.8674 - val_acc: 0.6875
Training took a total of 0:28:06.065400
```

3.3.2 Evaluate the Model

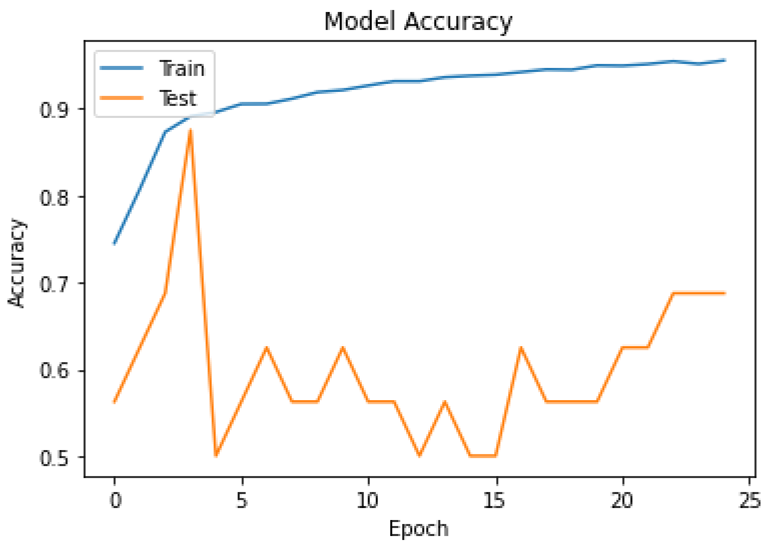
In [16]:

```
# evaluate model accuracy
scores = model.evaluate(test_gen)
print()
print('Model Accuracy: {}%'.format(scores[1]*100))
print('Model Loss: {}%'.format(scores[0]))
```

```
1/1 [==============================] - 0s 997us/step - loss: 0.4278 - acc: 0.8622

Model Accuracy: 86.21794581413269%
Model Loss: 0.42779043316841125%
```

In [17]:

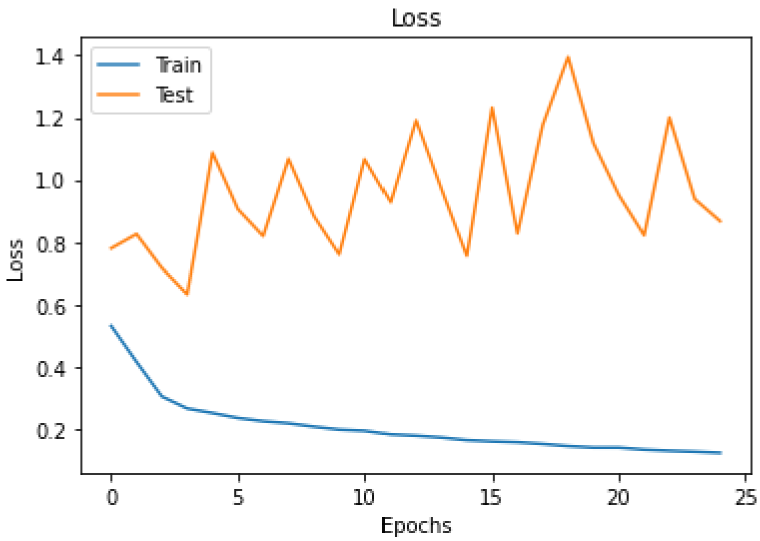
```
# model accuracy plot
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



- Notice lack of convergence between train and test set in terms of accuracy
- it looks like additional epochs did not have much effect on the testing accuracy but did improve training results.

In [18]:

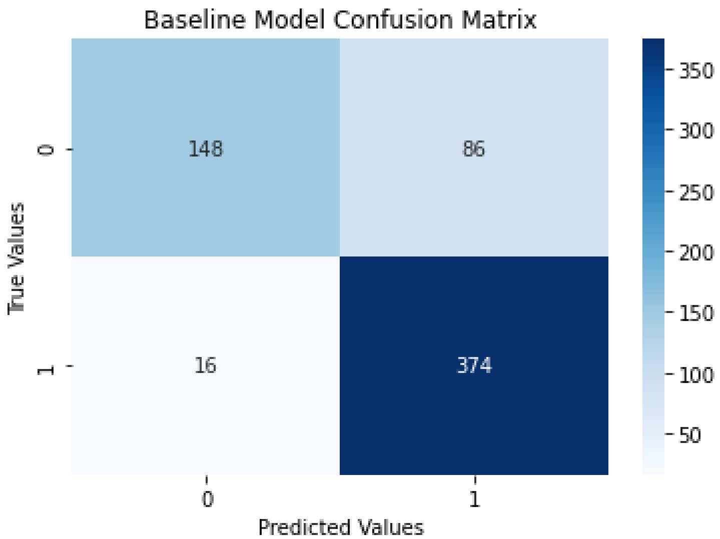
```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Train', 'Test'], loc='upper left')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



- Again, lack of convergence...
- It doesn't seem like additional epochs have an effect on validation loss.

In [19]:

```
# Create predictions for the model
y_hat_tmp = history.model.predict(X_test)

# classify y_hat as either 0 or 1 based on if val is < or >= to 0.5
thresh = 0.5
y_hat = (y_hat_tmp > thresh).astype(np.int)     # cast 0 or 1 to y_hat valu

y_t = y_test.astype(np.int)     # cast 0 or 1 to y_test values

cm_vals = confusion_matrix(y_t, y_hat)     # get confusion matrix values

# plot confusion matrix values
sns.heatmap(
    cm_vals,
    annot=True,
    cmap='Blues',
    fmt='0.5g'
)

plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Baseline Model Confusion Matrix')
plt.show()
```



The false positive rate is fairly high. These are patients that do not have pneumonia, but were predicted as having the disease. There were not many false negatives, which is good because it is better to overdiagnose than underdiagnose this disease because people that are not diagnosed with the disease but actually have it are more likely to suffer more from the disease because they will not receive immediate treatment.

In [56]:

```
sensitivity = 374 / (374+16)
ppv = 374 / (374+86)
specificity = 148 / (148+86)

print(sensitivity, ppv, specificity)
```

```
0.958974358974359 0.8130434782608695 0.6324786324786325
```

- Sensitivity is much better than the radiology residents.
- Positive Predictive Value is 81.3%, which is about 7% better than the residents.
- Specificity is over 40% worse in this model compared to the residents.
- The model did not produce many false negatives, which is quite good!

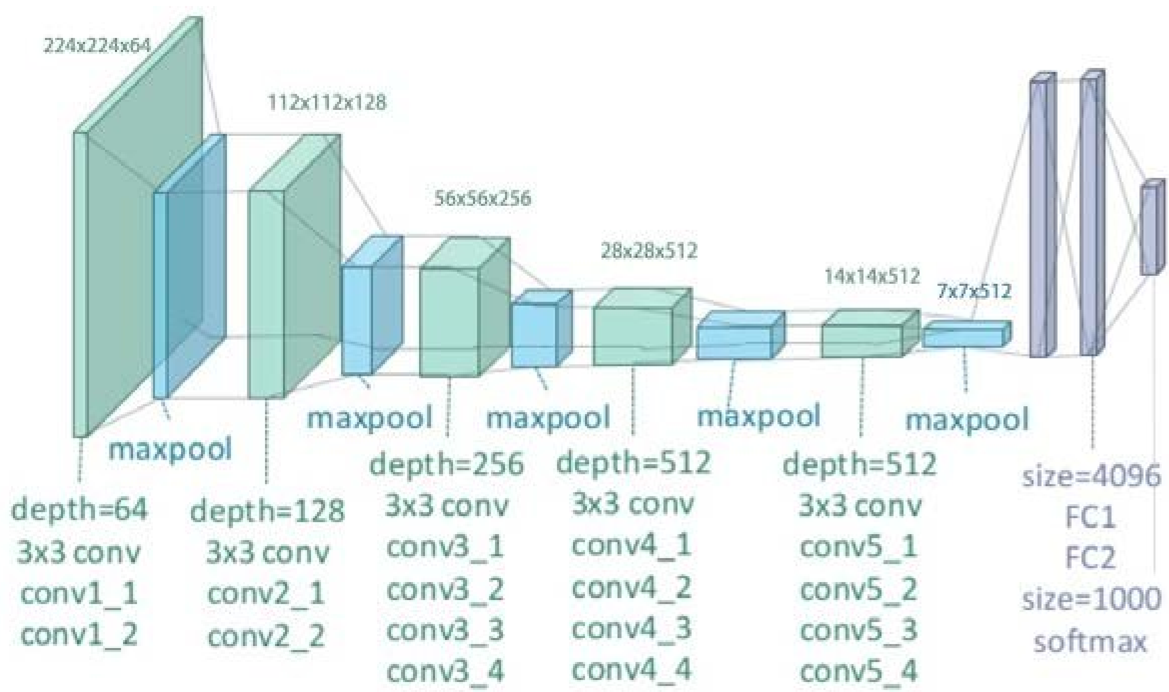
In [59]:
```
# define variables for recall and precision
r = sensitivity     # recall is the same as sensitivity...
p = ppv     # ppv is the same as precision...

F1_score = 2*((p*r)/(p+r))     # 2*((precision*recall) / (precision + recal
F1_score
```

Out[59]: 0.88

F1 score is better than CheXNet (Stanford University pneumonia prediction algorithm). More about this algorithm is mentioned in section 4.

3.4 Model 2: VGG-19 (Transfer Learning)

From [MathWork's description (https://www.mathworks.com/help/deeplearning/ref/vgg19.html)](https://www.mathworks.com/help/deeplearning/ref/vgg19.html): "VGG-19 is a convolutional neural network that is 19 layers deep. You can load a pretrained version of the network trained on more than a million images from the ImageNet database". Additionally, [this article (https://iq.opengenus.org/vgg19-architecture/)](https://iq.opengenus.org/vgg19-architecture/) explains the architecture of VGG-19 more fully.



This image shows the architecture of the VGG-19 model. The idea of transfer learning basically entails that a model that has been used to train a (very large) dataset can be utilized with other datasets because its architecture allows for a general enough approach for many different problems.

3.4.1 Create and fit model

In [21]:
```
# instantiate a VGG19 parameters w/ pre-determined weights from imagenet
vgg_params = VGG19(
    weights='imagenet',
    include_top=True
)

vgg_params.trainable = False

mod2 = models.Sequential()
mod2.add(vgg_params)
mod2.add(Flatten())
mod2.add(Dense(64, activation='relu'))
mod2.add(Dense(1, activation='sigmoid'))
```

In [22]:
```
# Compile the model
mod2.compile(
        loss='binary_crossentropy',
        optimizer='RMSprop',
        metrics=['acc']
)
```

The optimizing function has been changed to RMSprop instead of SGD. This function takes into account the average of the square of gradients and divides the gradient by the root of this average.

In [23]:
```
mod2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 1000)              143667240
_________________________________________________________________
flatten_1 (Flatten)          (None, 1000)              0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                64064
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 143,731,369
Trainable params: 64,129
Non-trainable params: 143,667,240
_________________________________________________________________
```

In [24]:

```
# fit the model and track time taken to train the model
original_start = datetime.datetime.now()
start = datetime.datetime.now()

history = mod2.fit(
    X_train,
    y_train,
    epochs=20,
    batch_size=16,
    validation_data=(X_val, y_val)
)

end = datetime.datetime.now()
elapsed = end - start
print('Training took a total of {}'.format(elapsed))
```

```
Epoch 1/20
326/326 [==============================] - 279s 855ms/step - loss: 0.5823 - a
cc: 0.7410 - val_loss: 0.8398 - val_acc: 0.5000
Epoch 2/20
326/326 [==============================] - 276s 845ms/step - loss: 0.5561 - a
cc: 0.7429 - val_loss: 0.8339 - val_acc: 0.5000
Epoch 3/20
326/326 [==============================] - 281s 861ms/step - loss: 0.5411 - a
cc: 0.7429 - val_loss: 0.7906 - val_acc: 0.5000
Epoch 4/20
326/326 [==============================] - 286s 876ms/step - loss: 0.5190 - a
cc: 0.7429 - val_loss: 0.7906 - val_acc: 0.5000
Epoch 5/20
326/326 [==============================] - 282s 864ms/step - loss: 0.4867 - a
cc: 0.7429 - val_loss: 0.7911 - val_acc: 0.5000
Epoch 6/20
326/326 [==============================] - 280s 858ms/step - loss: 0.4476 - a
cc: 0.7450 - val_loss: 0.7402 - val_acc: 0.5000
Epoch 7/20
326/326 [==============================] - 278s 854ms/step - loss: 0.4144 - a
cc: 0.7793 - val_loss: 0.8050 - val_acc: 0.5000
Epoch 8/20
326/326 [==============================] - 278s 853ms/step - loss: 0.3919 - a
cc: 0.8043 - val_loss: 0.8014 - val_acc: 0.5000
Epoch 9/20
326/326 [==============================] - 275s 842ms/step - loss: 0.3767 - a
cc: 0.8181 - val_loss: 0.7528 - val_acc: 0.5625
Epoch 10/20
326/326 [==============================] - 275s 843ms/step - loss: 0.3662 - a
cc: 0.8232 - val_loss: 0.8690 - val_acc: 0.5625
Epoch 11/20
326/326 [==============================] - 275s 843ms/step - loss: 0.3575 - a
cc: 0.8322 - val_loss: 0.8421 - val_acc: 0.5625
Epoch 12/20
326/326 [==============================] - 275s 842ms/step - loss: 0.3518 - a
cc: 0.8351 - val_loss: 0.8754 - val_acc: 0.5625
Epoch 13/20
326/326 [==============================] - 275s 843ms/step - loss: 0.3473 - a
cc: 0.8407 - val_loss: 0.9040 - val_acc: 0.5625
Epoch 14/20
```

```
326/326 [==============================] - 275s 842ms/step - loss: 0.3439 - a
cc: 0.8407 - val_loss: 0.9255 - val_acc: 0.5625
Epoch 15/20
326/326 [==============================] - 275s 843ms/step - loss: 0.3403 - a
cc: 0.8422 - val_loss: 0.8624 - val_acc: 0.5625
Epoch 16/20
326/326 [==============================] - 275s 843ms/step - loss: 0.3381 - a
cc: 0.8436 - val_loss: 0.9896 - val_acc: 0.5625
Epoch 17/20
326/326 [==============================] - 275s 843ms/step - loss: 0.3369 - a
cc: 0.8455 - val_loss: 0.8784 - val_acc: 0.5000
Epoch 18/20
326/326 [==============================] - 275s 842ms/step - loss: 0.3346 - a
cc: 0.8451 - val_loss: 0.9991 - val_acc: 0.5625
Epoch 19/20
326/326 [==============================] - 275s 842ms/step - loss: 0.3332 - a
cc: 0.8441 - val_loss: 0.9565 - val_acc: 0.5625
Epoch 20/20
326/326 [==============================] - 274s 841ms/step - loss: 0.3316 - a
cc: 0.8491 - val_loss: 0.9274 - val_acc: 0.5625
Training took a total of 1:32:31.962364
```

3.4.2 Evaluate the model

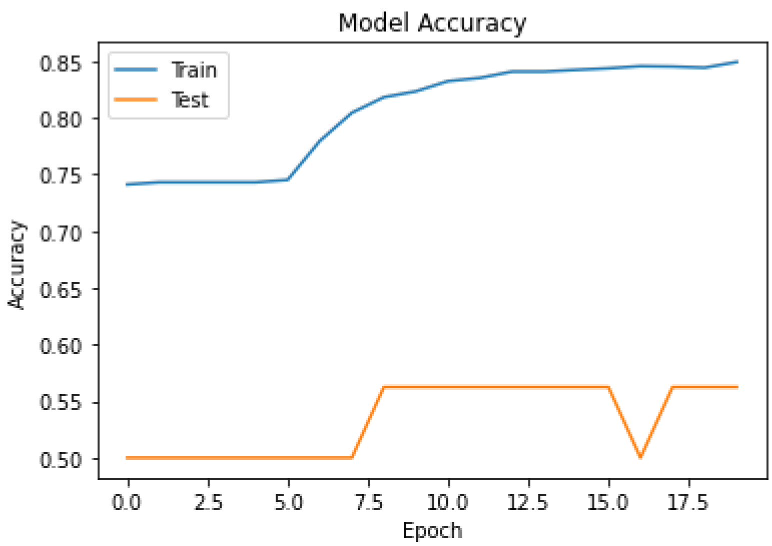
In [25]:

```
test_loss, test_acc = history.model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss}')
print(f'Test Acc: {test_acc}')
```

```
20/20 [==============================] - 31s 2s/step - loss: 0.5708 - acc: 0.73
88
Test Loss: 0.5708346962928772
Test Acc: 0.7387820482254028
```

In [26]:

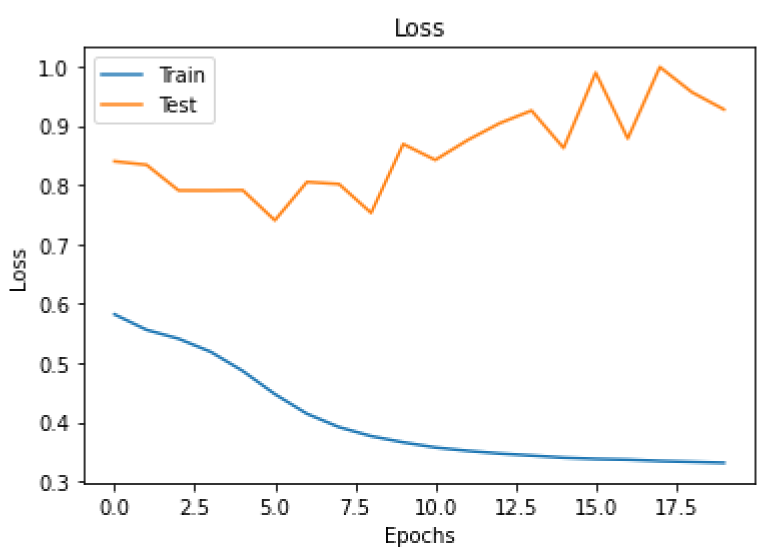
```
# model accuracy plot
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Accuracy of test set flattens quickly. Non-convergence between test and train sets.

In [27]:

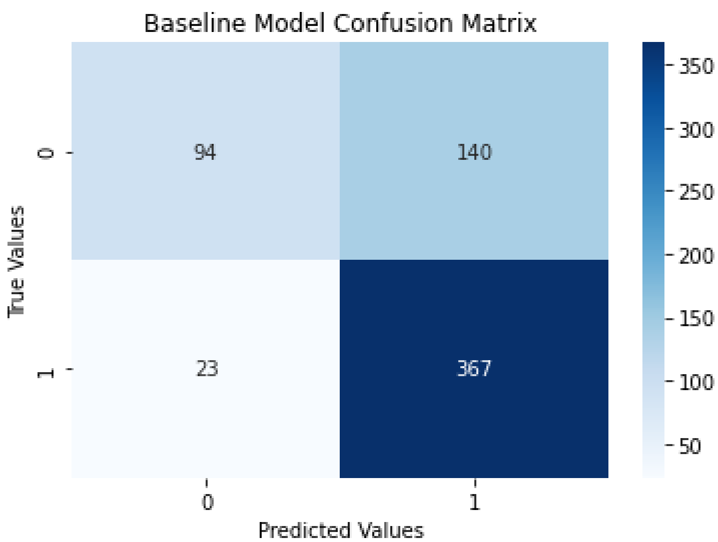
```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Train', 'Test'], loc='upper left')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



Loss of test set increases with more epochs... that's not a good sign for this model.

In [28]:

```
# Create predictions for the model
y_hat_tmp = history.model.predict(X_test)

# classify y_hat as either 0 or 1 based on if val is < or >= to 0.5
thresh = 0.5
y_hat = (y_hat_tmp > thresh).astype(np.int)     # cast 0 or 1 to y_hat valu

y_t = y_test.astype(np.int)     # cast 0 or 1 to y_test values

cm_vals = confusion_matrix(y_t, y_hat)     # get confusion matrix values

# plot confusion matrix values
sns.heatmap(
    cm_vals,
    annot=True,
    cmap='Blues',
    fmt='0.5g'
)

plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Baseline Model Confusion Matrix')
plt.show()
```



There is a significant amount of false positives.

In [29]:

```
print(classification_report(y_t, y_hat))
```

```
              precision    recall  f1-score   support

           0       0.80      0.40      0.54       234
           1       0.72      0.94      0.82       390

    accuracy                           0.74       624
   macro avg       0.76      0.67      0.68       624
weighted avg       0.75      0.74      0.71       624
```

There are quite a few false positives, but not many false negatives.

3.5 Model 3: VGG-19

Reducing the size of the training set to improve speed of the model. This reduction will also balance class distribution (remove imbalance).

3.5.1 Subsample Training Images and Pre-Process

In [30]:

```
# use df_train to randomly select 250 images with 0 and 1 from pneumonia co
df_tmp_norm = df_train[df_train.pneumonia==0]
df_tmp_n = df_tmp_norm.sample(n=250, random_state=42)
df_tmp_pn = df_train[df_train.pneumonia==1]
df_tmp_pn1 = df_tmp_pn.sample(n=250, random_state=42)

# concatenate dfs and then randomize
df_t3 = pd.concat([df_tmp_n, df_tmp_pn1])
df_t3 = df_t3.sample(frac=1)     # randomize
df_t3 = df_t3.reset_index(drop=True)
df_t3
```

Out[30]:

	image	pneumonia
0	data\train\NORMAL\NORMAL2-IM-0995-0001.jpeg	0
1	data\train\NORMAL\IM-0363-0001.jpeg	0
2	data\train\PNEUMONIA\person433_bacteria_1876.jpeg	1
3	data\train\PNEUMONIA\person1609_bacteria_4236....	1
4	data\train\PNEUMONIA\person554_bacteria_2321.jpeg	1
...	...	...
495	data\train\PNEUMONIA\person639_virus_1220.jpeg	1
496	data\train\PNEUMONIA\person1290_bacteria_3253....	1
497	data\train\PNEUMONIA\person1078_bacteria_3018....	1
498	data\train\NORMAL\NORMAL2-IM-0803-0001.jpeg	0
499	data\train\PNEUMONIA\person1241_bacteria_3197....	1

500 rows × 2 columns

This dataframe is a subset from the training data and contains 250 images from each category (pneumonia and normal).

Now that the data is subsampled, the file paths for each image needs to be read as an image and stored as an array.

In [31]:
```
# PIL image processing
from PIL import Image

images = []

for index, row in df_t3.iterrows():
    path=row['image']
    image = load_img(path, grayscale=False, color_mode="rgb",
                     target_size=(224, 224), interpolation="nearest")
    img_arr = img_to_array(image)
    images.append(img_arr)
```

In [32]:
```
# cast x and y train (images and y_train3) to np.arrays in order to fit
images = np.array(images)     # new X_train
y_train3 = np.array(df_t3.pneumonia)     # new y_train
```

3.5.2 Create and fit model

In [33]:
```
# instantiate a VGG19 parameters w/ pre-determined weights from imagenet
vgg_params = VGG19(
    weights='imagenet',
    include_top=False,
    input_tensor=Input(shape=(224, 224, 3))
)

vgg_params.trainable = False

mod3 = models.Sequential()

mod3.add(vgg_params)
mod3.add(Flatten())
mod3.add(Dense(64, activation='relu'))
mod3.add(Dense(1, activation='sigmoid'))

mod3.compile(loss='binary_crossentropy',
                optimizer='RMSprop',
                metrics=['acc'])
```

In [34]:
```
1 mod3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
vgg19 (Functional)          (None, 7, 7, 512)         20024384

_________________________________________________________________
flatten_2 (Flatten)         (None, 25088)             0

_________________________________________________________________
dense_4 (Dense)             (None, 64)                1605696

_________________________________________________________________
dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 21,630,145
Trainable params: 1,605,761
Non-trainable params: 20,024,384
_________________________________________________________________
```

Large amount of non-trainable parameters because the model is using predetermined weights from imagenet.

In [35]:

```
original_start = datetime.datetime.now()
start = datetime.datetime.now()

# Generate augmented images
img_perms = ImageDataGenerator(
    rescale = 1. / 255,     # multiply the data by the value provided
    shear_range = 0.2,     # this distorts the image along an axis
    zoom_range = 0.2,     # range for random zoom
    horizontal_flip = True     # random horizontal flip
)

history = mod3.fit(
    img_perms.flow(images, y_train3, batch_size=32),
    epochs=25,
    validation_data=(X_val, y_val),
)

end = datetime.datetime.now()
elapsed = end - start
print('Training took a total of {}'.format(elapsed))
```

```
Epoch 1/25
16/16 [==============================] - 25s 2s/step - loss: 2.2709 - acc: 0.
6360 - val_loss: 0.5540 - val_acc: 0.6875
Epoch 2/25
16/16 [==============================] - 25s 2s/step - loss: 0.6476 - acc: 0.
7940 - val_loss: 2.0813 - val_acc: 0.6250
Epoch 3/25
16/16 [==============================] - 25s 2s/step - loss: 0.4586 - acc: 0.
8440 - val_loss: 0.4758 - val_acc: 0.7500
Epoch 4/25
16/16 [==============================] - 25s 2s/step - loss: 0.5468 - acc: 0.
8280 - val_loss: 1.2742 - val_acc: 0.6875
Epoch 5/25
16/16 [==============================] - 25s 2s/step - loss: 0.4374 - acc: 0.
8420 - val_loss: 1.1504 - val_acc: 0.6250
Epoch 6/25
16/16 [==============================] - 25s 2s/step - loss: 0.4462 - acc: 0.
8640 - val_loss: 0.5962 - val_acc: 0.6250
Epoch 7/25
```

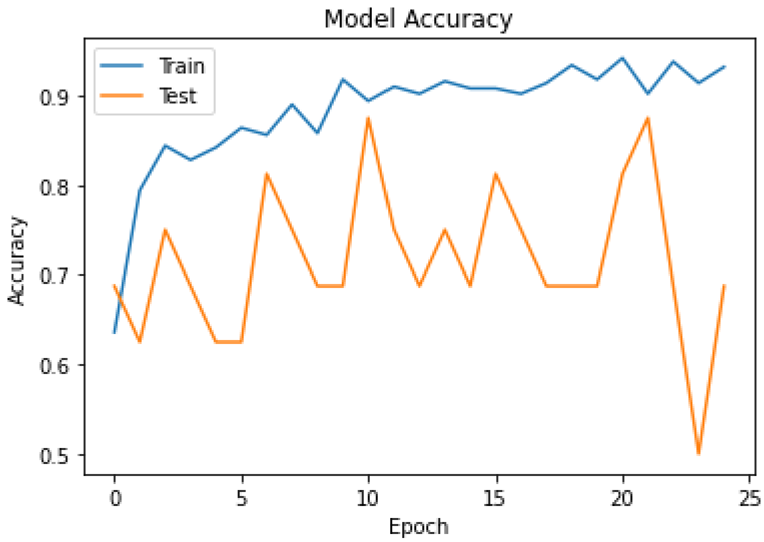
3.5.3 Evaluate the model

In [36]:

```
test_loss, test_acc = history.model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss}')
print(f'Test Acc: {test_acc}')
```

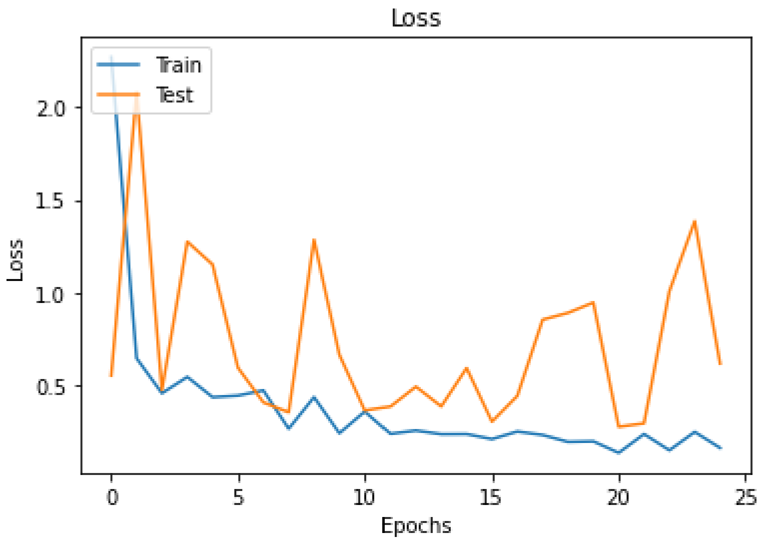
```
20/20 [==============================] - 31s 2s/step - loss: 0.3851 - acc: 0.86
06
Test Loss: 0.3850596845149994
Test Acc: 0.8605769276618958
```

In [37]:
```
# model accuracy plot
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



It seems like test set accuracy might be trending upwards, but it's difficult to tell with so few epochs. It's too volatile with this amount of epochs to understand the overall trend.

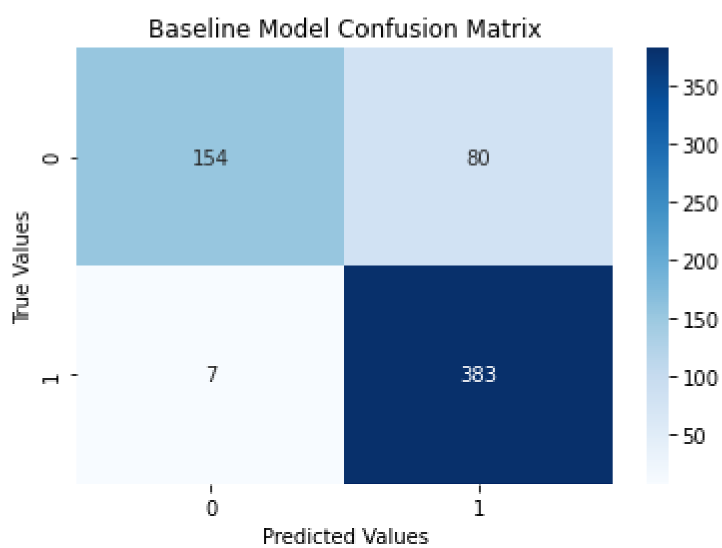
In [38]:
```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Train', 'Test'], loc='upper left')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



Loss is trending downwards for both train and test sets.

In [39]:

```
# Create predictions for the model
y_hat_tmp = history.model.predict(X_test)
# classify y_hat as either 0 or 1 based on if val is < or >= to 0.5
thresh = 0.5
y_hat = (y_hat_tmp > thresh).astype(np.int)     # cast 0 or 1 to y_hat val

y_t = y_test.astype(np.int)     # cast 0 or 1 to y_test values

cm_vals = confusion_matrix(y_t, y_hat)     # get confusion matrix values

# plot confusion matrix values
sns.heatmap(
    cm_vals,
    annot=True,
    cmap='Blues',
    fmt='0.5g'
)

plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Baseline Model Confusion Matrix')
plt.show()
```



Large improvement in terms of false positive rate from the baseline model. Low amount of false negatives, which is great.

3.6 Model 4: Dropout Regularization

Dropout Regularization is a technique for reducing overfitting and improve generalization (the ability for the model to make valuable predictions on a new set of data). Specifically, the dropout technique allows the model to emulate a very large model because it randomly discards nodes. The model will still have the number of layers originally created, but the nodes will randomly be thrown out. This method is useful for reducing compute intensity and incorporates an element of randomness that's effective for reducing overfitting.

3.6.1 Create and fit model

In [40]:

```
mod4 = models.Sequential()

mod4.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                    input_shape=(224, 224, 3)))
mod4.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
mod4.add(MaxPooling2D((2, 2)))

mod4.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
mod4.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))
mod4.add(MaxPooling2D(2, 2))

mod4.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
mod4.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
mod4.add(MaxPooling2D((2, 2)))

mod4.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
mod4.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
mod4.add(MaxPooling2D((2, 2)))

# add dropout regularization
mod4.add(layers.Flatten())
mod4.add(layers.Dense(512, activation='relu'))
mod4.add(layers.Dropout(0.3))
mod4.add(layers.Dense(512, activation='relu'))
mod4.add(layers.Dropout(0.3))
mod4.add(layers.Dense(1, activation='sigmoid'))

# Compile the model
mod4.compile(
    optimizer='RMSprop',
    loss='binary_crossentropy',
    metrics=['acc']
)

mod4.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)           (None, 222, 222, 32)      896
_________________________________________________________________
conv2d_4 (Conv2D)           (None, 220, 220, 32)      9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 110, 110, 32)     0
_________________________________________________________________
conv2d_5 (Conv2D)           (None, 108, 108, 32)      9248
_________________________________________________________________
conv2d_6 (Conv2D)           (None, 106, 106, 32)      9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 53, 53, 32)       0
_________________________________________________________________
conv2d_7 (Conv2D)           (None, 51, 51, 64)        18496
_________________________________________________________________
conv2d_8 (Conv2D)           (None, 49, 49, 64)        36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 24, 24, 64)       0
```

```
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 22, 22, 128)       73856     
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 20, 20, 128)       147584    
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 10, 10, 128)       0         
_________________________________________________________________
flatten_3 (Flatten)          (None, 12800)             0         
_________________________________________________________________
dense_6 (Dense)              (None, 512)               6554112   
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_7 (Dense)              (None, 512)               262656    
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 513       
=================================================================
Total params: 7,122,785
Trainable params: 7,122,785
Non-trainable params: 0
_________________________________________________________________
```

In [41]:

```
original_start = datetime.datetime.now()
start = datetime.datetime.now()

history = mod4.fit(
    X_train,
    y_train,
    validation_data=(X_test, y_test),
    epochs=20
)

end = datetime.datetime.now()
elapsed = end - start
print('Training took a total of {}'.format(elapsed))
```

```
Epoch 1/20
163/163 [==============================] - 174s 1s/step - loss: 0.5636 - acc:
0.7395 - val_loss: 0.6028 - val_acc: 0.6298
Epoch 2/20
163/163 [==============================] - 174s 1s/step - loss: 0.3777 - acc:
0.8434 - val_loss: 0.4184 - val_acc: 0.8221
Epoch 3/20
163/163 [==============================] - 173s 1s/step - loss: 0.2919 - acc:
0.8850 - val_loss: 0.5073 - val_acc: 0.8077
Epoch 4/20
163/163 [==============================] - 172s 1s/step - loss: 0.2467 - acc:
0.9030 - val_loss: 0.3518 - val_acc: 0.8429
Epoch 5/20
163/163 [==============================] - 162s 994ms/step - loss: 0.2095 - ac
c: 0.9185 - val_loss: 1.1602 - val_acc: 0.6971
Epoch 6/20
163/163 [==============================] - 159s 978ms/step - loss: 0.1880 - ac
c: 0.9268 - val_loss: 0.6727 - val_acc: 0.8397
Epoch 7/20
163/163 [==============================] - 162s 994ms/step - loss: 0.1829 - ac
c: 0.9310 - val_loss: 0.4201 - val_acc: 0.8702
Epoch 8/20
163/163 [==============================] - 161s 987ms/step - loss: 0.1632 - ac
c: 0.9388 - val_loss: 0.5920 - val_acc: 0.8365
Epoch 9/20
163/163 [==============================] - 160s 981ms/step - loss: 0.1470 - ac
c: 0.9452 - val_loss: 0.6668 - val_acc: 0.8221
Epoch 10/20
163/163 [==============================] - 160s 981ms/step - loss: 0.1317 - ac
c: 0.9530 - val_loss: 0.4503 - val_acc: 0.8542
Epoch 11/20
163/163 [==============================] - 159s 977ms/step - loss: 0.1234 - ac
c: 0.9548 - val_loss: 0.4713 - val_acc: 0.8718
Epoch 12/20
163/163 [==============================] - 161s 988ms/step - loss: 0.1078 - ac
c: 0.9617 - val_loss: 0.9840 - val_acc: 0.8189
Epoch 13/20
163/163 [==============================] - 160s 982ms/step - loss: 0.1102 - ac
c: 0.9626 - val_loss: 0.7071 - val_acc: 0.8173
Epoch 14/20
163/163 [==============================] - 160s 979ms/step - loss: 0.0951 - ac
c: 0.9659 - val_loss: 1.3957 - val_acc: 0.7708
Epoch 15/20
```

```
163/163 [==============================] - 162s 994ms/step - loss: 0.0944 - ac
c: 0.9693 - val_loss: 0.7211 - val_acc: 0.8365
Epoch 16/20
163/163 [==============================] - 162s 993ms/step - loss: 0.0842 - ac
c: 0.9686 - val_loss: 1.0488 - val_acc: 0.8157
Epoch 17/20
163/163 [==============================] - 162s 992ms/step - loss: 0.0794 - ac
c: 0.9722 - val_loss: 0.6730 - val_acc: 0.8429
Epoch 18/20
163/163 [==============================] - 160s 981ms/step - loss: 0.0716 - ac
c: 0.9783 - val_loss: 0.9884 - val_acc: 0.8157
Epoch 19/20
163/163 [==============================] - 159s 978ms/step - loss: 0.0689 - ac
c: 0.9770 - val_loss: 1.2660 - val_acc: 0.7853
Epoch 20/20
163/163 [==============================] - 161s 987ms/step - loss: 0.0594 - ac
c: 0.9793 - val_loss: 1.2375 - val_acc: 0.8093
Training took a total of 0:54:44.851741
```

3.6.2 Evaluate the model

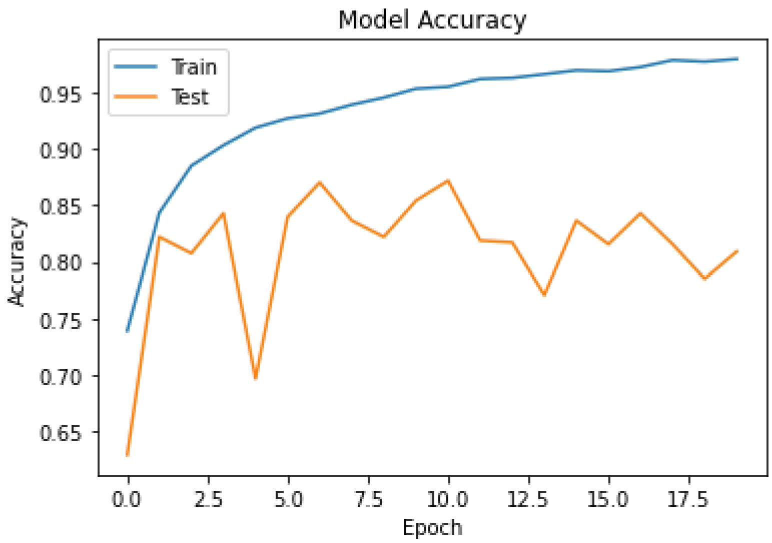
In [42]:
```
test_loss, test_acc = history.model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss}')
print(f'Test Acc: {test_acc}')
```

```
20/20 [==============================] - 3s 135ms/step - loss: 1.2375 - acc: 0.
8093
Test Loss: 1.237475037574768
Test Acc: 0.8092948794364929
```

The accuracy in this model is a bit better than the baseline model but worse than model 3.

In [43]:

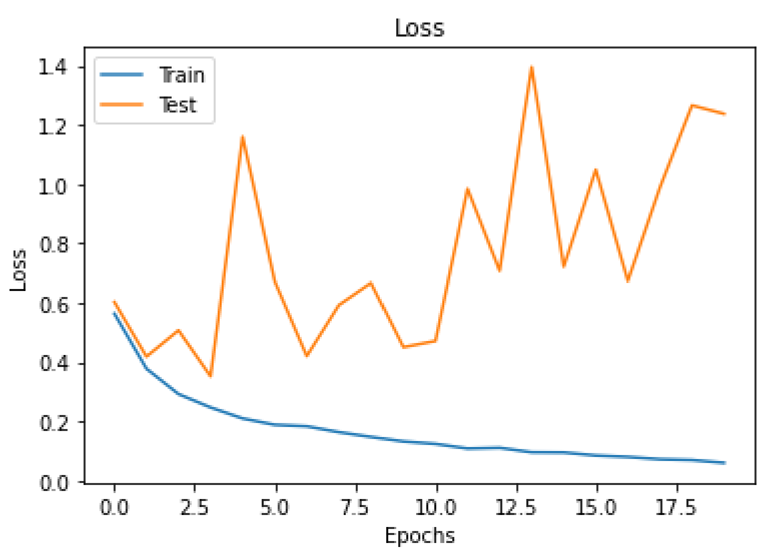
```
# model accuracy plot
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Accuracy still trending upwards at the end of fitting the model. It might be worthwhile to use more epochs for this model.

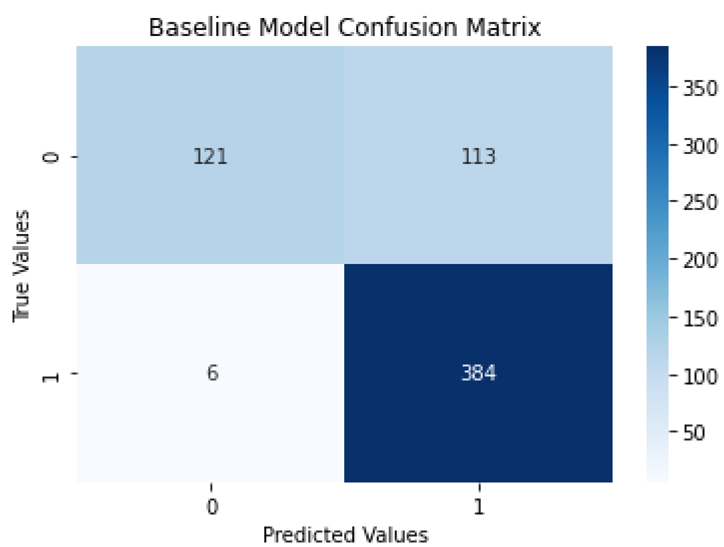
In [44]:

```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Train', 'Test'], loc='upper left')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



Loss looks flat for the test set, but it could be worthwhile to utilize more epochs to understand a larger trend.

In [45]:
```
# Create predictions for the model
y_hat_tmp = history.model.predict(X_test)
# classify y_hat as either 0 or 1 based on if val is < or >= to 0.5
thresh = 0.5
y_hat = (y_hat_tmp > thresh).astype(np.int)     # cast 0 or 1 to y_hat valu

y_t = y_test.astype(np.int)     # cast 0 or 1 to y_test values

cm_vals = confusion_matrix(y_t, y_hat)     # get confusion matrix values

# plot confusion matrix values
sns.heatmap(
    cm_vals,
    annot=True,
    cmap='Blues',
    fmt='0.5g'
)

plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Baseline Model Confusion Matrix')
plt.show()
```



Slightly worse results than model 3, but decent as far as a comparison to the baseline model.

3.7 Model 5: Optimize Best Model

This model will try to optimize the best model so far and produce a better model.

Model 3 performed the best in terms of the metrics this project is interested in optimizing.

3.7.1 Create and fit model

In [46]:
```
original_start = datetime.datetime.now()
start = datetime.datetime.now()

history = mod3.fit(
    img_perms.flow(images, y_train3, batch_size=32),
    epochs=250,     # 250 epochs should take about 5 hours to run
    batch_size=32,
    validation_data=(X_val, y_val)
)

end = datetime.datetime.now()
elapsed = end - start
print('Training took a total of {}'.format(elapsed))
```

```
Epoch 125/250
16/16 [==============================] - 25s 2s/step - loss: 0.0163 - acc: 0.
9960 - val_loss: 0.2002 - val_acc: 0.8750
Epoch 126/250
16/16 [==============================] - 25s 2s/step - loss: 0.0333 - acc: 0.
9900 - val_loss: 0.0988 - val_acc: 0.9375
Epoch 127/250
16/16 [==============================] - 25s 2s/step - loss: 0.0269 - acc: 0.
9900 - val_loss: 0.8446 - val_acc: 0.7500
Epoch 128/250
16/16 [==============================] - 25s 2s/step - loss: 0.0989 - acc: 0.
```

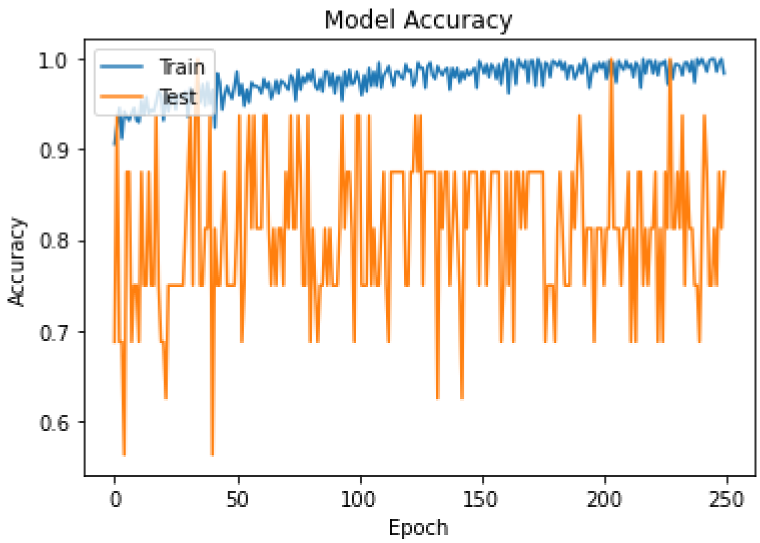
3.7.2 Evaluate the model

In [47]:
```
test_loss, test_acc = history.model.evaluate(X_test, y_test)
print(f'Test Loss: {test_loss}')
print(f'Test Acc: {test_acc}')
```

```
20/20 [==============================] - 30s 2s/step - loss: 0.6901 - acc: 0.89
42
Test Loss: 0.6900997161865234
Test Acc: 0.8942307829856873
```

In [48]:

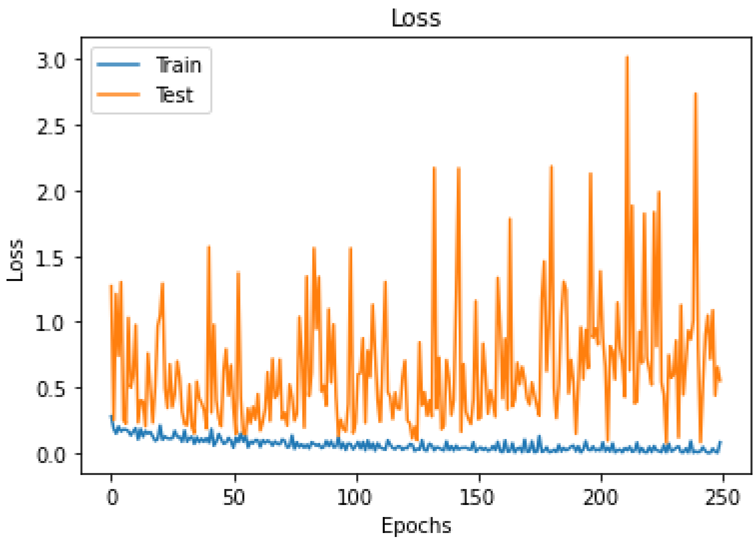
```
# model accuracy plot
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Model accuracy seems to generally trend upwards in the train set and somewhat in the test set.

In [49]:

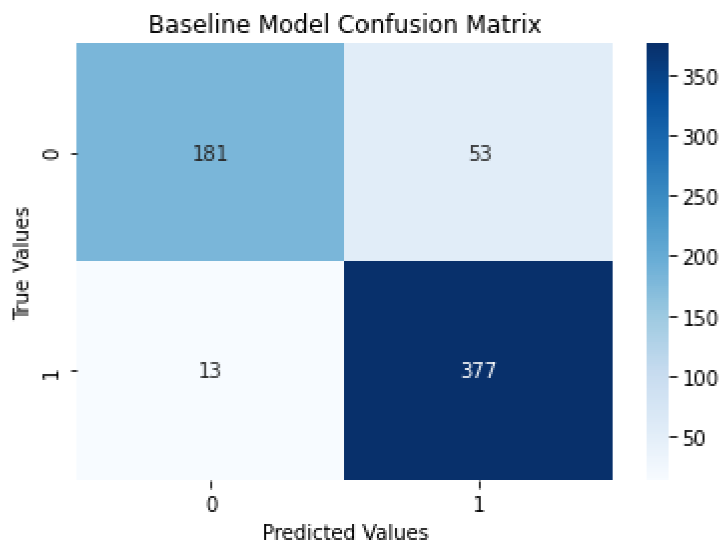
```
plt.figure()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Train', 'Test'], loc='upper left')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()
```



Loss looks pretty volatile for the test set, but there may be a significant trend down.

In [50]:

```
# Create predictions for the model
y_hat_tmp = history.model.predict(X_test)
# classify y_hat as either 0 or 1 based on if val is < or >= to 0.5
thresh = 0.5
y_hat = (y_hat_tmp > thresh).astype(np.int)     # cast 0 or 1 to y_hat valu

y_t = y_test.astype(np.int)     # cast 0 or 1 to y_test values

cm_vals = confusion_matrix(y_t, y_hat)     # get confusion matrix values

# plot confusion matrix values
sns.heatmap(
    cm_vals,
    annot=True,
    cmap='Blues',
    fmt='0.5g'
)

plt.xlabel('Predicted Values')
plt.ylabel('True Values')
plt.title('Baseline Model Confusion Matrix')
plt.show()
```



Significantly less false positives than any other model, but there is an increase in false negatives compared to the other models. The accuracy in this model is better than any other so far.

4 Intepretation of final model

Recall how results of this model are being evaluated: According to IBM, the sensitivity, specificity, and positive predictive value for radiology residents is as follows: 0.720, 0.973, and 0.682 respectively. How did the results of my final model compare with these results?

In [54]:
```
sensitivity = 377 / (377+53)
ppv = 377 / (377+13)
specificity = 181 / (181+13)

print(sensitivity, ppv, specificity)
```

```
0.8767441860465116 0.9666666666666667 0.9329896907216495
```

In comparison to the radiology residents, this model's sensitivity is better by about 16%, specificity is worse by 4%, and ppv is better by about 30%. Even though my model performed slightly worse in terms of specificity, it did perform much better in terms of sensitivity and positive predictive value. Overall, it seems like my model does better than the radiology residents in the IBM article, but I want another metric to evaluate it with.

I will also compare the F1 score of this model to some F1 scores found among radiology professionals. F1 seems like an appropriate metric because it takes into account penalization factors for both false negatives and false positives. The formula can be reviewed below:

$$F1\ score = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

One of the first x-ray pneumonia-predicting algorithms was CheXNet, which was developed by Andrew Ng and others at Stanford University. This algorithm purported a better F1 score than the average of several radiologists at Stanford. Their algorithm had an F1 score of 0.435. This score is better in comparison to the radiologists' averaged F1 score of 0.387. More information on the specifics of the study and algorithm can be found [here (https://arxiv.org/pdf/1711.05225.pdf)](https://arxiv.org/pdf/1711.05225.pdf).

In [55]:
```
# define variables for recall and precision
r = sensitivity     # recall is the same as sensitivity...
p = ppv     # ppv is the same as precision...

F1_score = 2*((p*r)/(p+r))     # 2*((precision*recall) / (precision + recal
F1_score
```

Out[55]:
```
0.9195121951219511
```

The F1 score for my model is about .925, which is much better than the average F1 score of the radiologists cited in the Stanford paper.